An Attempt at Explicating the Relationship between Knowledge, Systems and Engineering

Amar Banerjee
Tata Consultancy Services Research
Pune
amar.banerjee@tcs.com

Venkatesh Choppella
International Institute of Information Technology
Hyderabad
venkatesh.choppella@iiit.ac.in

Viswanath Kasturi
International Institute of Information Technology
Hyderabad
viswanath.iiithyd@gmail.com

Swaminathan Natarajan
Tata Consultancy Services Research
Chennai
swami.n@tcs.com

Padmalata V. Nistala
Tata Consultancy Services Research
Hyderabad
nistala.padma@tcs.com

Kesav Nori
International Institute of Information Technology
Hyderabad
kesav.nori@gmail.com

ABSTRACT

Software systems often serve as the agents of operation for both enterprise systems and embedded systems. Engineering such systems is a knowledge-centric activity. A clear understanding of the relationship between knowledge, systems and engineering can help us to establish firm theoretical foundations for software and systems engineering.

Currently we have a strong intuitive understanding of how knowledge flows into engineering, while our understanding of the relationship between systems and knowledge is part explicit and part tacit. A symptom of this is that we have difficulty in building unified models of large systems such as telescopes and enterprises that span multiple knowledge domains and viewpoints. We are able to build multiple models covering various aspects and particular viewpoints. However, we have challenges in integrating them into a single unified model. Another symptom is that software and systems engineering practice are widely viewed as empirical fields, without sufficiently strong theoretical foundations.

This work attempts to explicate and synthesize our common intuitive understanding in this space to develop a conceptual model of the relationships. It then explores the validity of this model by examining the extent to which it is able to explain and illuminate current engineering practices and issues. This is an initial strawman version of the model, presented with a view to obtaining feedback and inputs from the community.

CCS CONCEPTS

• **Theory of computation → Semantics and reasoning**; • **Mathematics of computing** → *Information theory*;

KEYWORDS

knowledge-centric explication of engineering, relationship between knowledge and systems, conceptual model of systems engineering, theory of software systems engineering, viewpoint mappings

ACM Reference Format:
Amar Banerjee, Venkatesh Choppella, Viswanath Kasturi, Swaminathan Natarajan, Padmalata V. Nistala, and Kesav Nori. 2018. An Attempt at Explicating the Relationship between Knowledge, Systems and Engineering. In *ISEC '18: Innovations in Software Engineering Conference, February 9–11, 2018, Hyderabad, India.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3172871.3172875

1 INTRODUCTION

Software has been instrumental in explicating the logic of various domains, not only enterprise application domains such as banking and insurance, but also engineering domains such as infrastructure and operations. Given that software also increasingly serves as the agent of operations for a wide variety of systems ranging from photocopiers and radiotelescopes to enterprises, this has also included understanding and explicating different types of relationships among entities across domains.

Under the circumstances, it would be reasonable to expect that over time clarity would have emerged about knowledge structures within each domain, as well as about how these come together to form systems. In fact, we do observe that in domain after domain, there are efforts afoot to develop standard vocabularies and ontologies, to facilitate interoperation across tools and solutions for the domain. But when it comes to integrating tools across domains, or integrating domain-specific solution models to form an integrated system model, we encounter significant challenges, and end up with multiple silo-ed models, with only some people understanding how these all fit together. The best we seem to be able to do is vendor-specific integrations across tools.

A similar situation obtains in the area of software and systems engineering tools. While these are domains in their own right, it is also the case that the nature of engineering in these domains is that it involves understanding and working with multiple other domains, including both application domains such as radiotelescopes and insurance, quality domains such as performance and security, and activity domains such as risk management and operations. In order to truly support the knowledge-centric activity of engineering, we would like our tools to be able to work with knowledge structures in these target domains, but again we encounter significant challenges. Our tools understand the software or systems engineering domain, but not the target domain. The best we seem to be able to do is to build domain-specific tools that understand a particular target domain as well as the nature of software/systems.

To go beyond this situation, it is clear that we need a deeper, fundamental understanding of the relationship between knowledge, systems and engineering. This work is an inquiry into that space. Our starting point assumption is that there is probably no need to develop "novel" ideas, that over decades of software and systems engineering (in fact, thousands of years of systems engineering), we have all built up very strong intuitions and tacit understanding of how it all fits together. What is needed is to explicate and synthesize this understanding into a unified model of how engineering pulls together knowledge from multiple domains to generate systems.

The approach we take in our work is to separate the problems of conceptualizing the model, and investigating its validity. First, we postulate a model (currently a very early strawman version) based primarily on our intuitive understanding and experience, and of course the existing body of knowledge (to the extent we are aware of it). Then, we take a utilitarian approach to investigating its validity, by examining whether the model is consistent with our observations as engineers, whether it is able to expose (and preferably provide insight into) the practical issues we encounter in engineering, and whether it is able to provide us actionable insights into how we can overcome the challenges in providing knowledge-centric support for engineering solutions that span multiple domains.

The paper first explores some of the challenges in building integrated models of systems, and some of the related work. Next, we incrementally build up the conceptual model by making a series of observations about the various spaces involved. This is followed by an examination of its validity, in terms of how it explains and provides insight into some key engineering issues and practices, and the guidance it provides on how we can try to overcome the challenges by taking a knowledge-centric approach to integrated modeling and engineering. We conclude by assessing where we are currently, and the way forward.

2 MOTIVATION, CHALLENGES AND RELATED WORK

Our attention was focused on the problems of tool integration by our observations while participating in Square Kilometer Array, an international radiotelescope project. Currently, the project is in the design phase, involving about 500 engineers and scientists worldwide working on the design of various subsystems: receiver design, signal processing, scientific image processing, control, networking, physical ifnrastructure etc. The project uses state-of-the-art model-based engineering approaches: there is an overall system model using SysML, complemented by the use of domain-specific modelling tools for each area: CAD/CAM models, MATLAB models, network design models, physical layout models etc. However, these models cannot be integrated easily: the tools do not (in general) interoperate.

The impact of this can be understood at two levels. The obvious impact is that at the end of three years of design and such extensive modelling, there will be islands of design models, and the overall design needs to be understood from documentation. Moreover, the models will be hard to keep mutually consistent.

But the deeper impact can be seen if we take a closer look at the design phase itself. Each of the engineers working on the project is highly experienced, and have been involved in building other similar radiotelescopes. They have the solutions to the key design issues clearly in mind, and in fact, once the problem requirements and boundaries are fully defined, can develop the detailed solution in a matter of months i.e. less than 20-30% of the total. But the boundaries of the problem for each subsystem depend on the decisions made in every other subsystem, and the overall telescope requirements are constrained by budget and what is collectively achievable by all the subsystems acting together, within available budget. In effect, the project is a huge network of tens of thousands of decisions, about requirements, design choices, technology choices, interfaces, sizing, quality parameters etc. In effect, the primary challenge of the project is to achieve consistency across this entire network of thousands of decisions, and the practices involved in achieving this, including documentation, ICDs (interface control documents), version management, coordination meetings, iterative rework etc consume the remaining 70% of the project design effort. Just imagine how much of this effort could be saved if we had an integrated model and toolset spanning all of the domains involved, that could model and monitor consistency relationships across the entire network, and facilitate analysis of collective properties!

Clearly, there are large financial and technical incentives to build such integrated models that span multiple domains, given that similar challenges are faced by any large project, and also large enterprises that spans multiple domains and viewpoints. Yet the problem has not been solved over several decades. Obviously there must be deep challenges that make it difficult to integrate models. Let us examine some of them:

- **Vocabulary and concepts**: The root of the problem is that each domain has its own vocabulary and concepts. An antenna may be described as a receiver in signal processing, as a dish in the physical viewpoint, and as a load in the power viewpoint. The mappings between these concepts depends on the specific system and context, so it is not a simple matter of establishing translations across vocabularies directly at the level of domains.
- **Missing semantics**: The SysML model, which has a view of the entire system, could potentially bridge the gap, since entities, attributes and operations in every other viewpoint could be mapped to the corresponding specific system elements. But the SysML model is itself captured in terms of generic systems concepts, such as entities and attributes. It does not have the semantic information to distinguish between an antenna and (say) a network router. So users must do all the hard work to link every element in every other domain model to the corresponding system model element, either manually or through naming conventions, which is

tedious, error-prone and fragile in the presence of constant evolution of all the models - unworkable, really. For this reason, integration is typically limited to tools produced by the same vendor, where the process of model creation itself can establish identity of elements across viewpoints.
- **Loss of information**: The situation is further complicated by the fact that there may not be a one-to-one relationship between elements across models. For example, a site may have multiple entities such as dishes and beamformers, but a power model or network model may combine them into a single load. In general, there may be many-to-many relationships between elements in one model and elements in another model.

It can be seen from the above that if we want to solve the problem of generating integrated models across domains and viewpoints, we need fundamental insights into the nature of the relationships between systems and knowledge domains.

The situation with engineering is that we have a strong body of practice knowledge, but there is a sense that it is all empirical, that adequate theoretical foundations are lacking. We see one particular gap that may be contributing to this: we all know that software and systems engineering is intensive in domain knowledge, but there is very little in our current theory and practice that explicitly discusses the role of domain knowledge in engineering. This is the gap that we aim to fill.

Our work draws strongly upon the standard body of knowledge in systems (SEBoK) and software (SWEBoK), and upon the systems concepts embodied in modelling approaches such as SysML and IDEF. There is considerable work on the challenges in integrating systems viewpoints, including the use of ontological approaches [2], [5]. There is also considerable work on a semantic view of the nature of engineering, such as [1] that explicates the nature of relationships between blocks in systems. The relationship between the model world and real world was explored in [6], while [4] developed the idea that engineering activity involves producing a set of models with mutual consistency relationships. Our work draws upon these and a great deal of other such work in the field that has provided deep conceptual insights into the nature of systems, engineering and knowledge, and the relationship among them. In short, the ideas proposed in this work are not new, our contribution is to synthesize insights from many sources into a unified conceptual framework.

3 CONCEPTUAL MODEL

This section incrementally builds up a conceptual model by making a series of observations about knowledge, systems and engineering. These observations are based on common intuitive understanding prevalent among engineers and ideas from the standard bodies of knowledge. As such, no individual observation is novel: it is only the synthesis that may possibly be novel. After building up the conceptual model, in the next section we will explore its validity in terms of its power to explain common practices and issues in engineering.

3.1 The form of knowledge

We are interested in knowledge from the viewpoint of systems and their engineering, so we start by exploring the form of engineering and scientific knowledge.

Our first observation is that the knowledge of primary interest to engineers and scientists is that which expresses relationships between structures, processes and outcomes [3]. Scientists are interested in phenomena, and explaining the phenomena in terms of the processes that operate over structures to generate the phenomena. (Note: in some cases the knowledge may relate purely to structural relationships e.g. equations and constraints among attributes of entities, or purely to process inter-relationships e.g. complex behaviour and emergence). Engineers are interested in knowledge that indicates how to achieve desired outcomes, in terms of the required structural and process elements. Thus in both cases, the resulting knowledge is a relationship between structures, processes and outcomes. The knowledge may have applicability qualifiers in terms of contextual factors and assumptions.

Of course, in addition to such *patterns* knowledge, there is also the ontological knowledge on which it is based:

- Structure is expressed in terms of entities, their attributes, relationships among entities and relationships among attributes.
- Process is expressed in terms of sequences of operations on entities.
- Outcomes capture behavioural aspects, and are expressed in terms of characterizations of the behaviour (including functions, associated features, services that capture logical groupings of interactional capabilities between a system and its context, and quality characteristics that capture particular aspects of behaviour) as well as characterizations of the structures and processes that give rise to behaviour.

We also have situational knowledge: facts about the world of interest, including both transient information (e.g. set of resources available at a given point in time), and characterizations of the world (e.g. typical stakeholder concerns). As we shall see next, there are other kinds of knowledge as well, including concepts at the level of knowledge domains, and relationships among knowledge domains.

While all these types of knowledge exist, we can view them as either facilitating or complementing the core patterns knowledge. Hence, our first observation is that *Systems knowledge is primarily in the form of patterns that express relationships between structures, processes and outcomes, qualified by context.*

3.2 Knowledge domains

We organize knowledge items (patterns, ontological knowledge, situational knowledge etc) into knowledge domains. A knowledge domain is a collection of related knowledge items: the patterns are expressed over the same *vocabulary* of structure, process and outcome elements i.e. the same set of entities, attributes, relationships, operations, functions, quality characteristics etc.

A knowledge domain may also include additional concepts e.g. systems engineering knowledge introduces concepts such as engineering life cycle, stages and phases. These characterize and organize the set of knowledge items, enabling the expression of relationships among them, and eventually facilitating the development of synthesis and analysis theories in the domain. Such theories enable reasoning about the compositional result of multiple patterns i.e. the relationships between configurations (structures), the collection of processes enabled by the configurations, and the outcomes produced.

It should be noted that knowledge items may include explicit but informal knowledge such as guidelines and thumb rules, and empirically determined practices. Thus knowledge domains may span the range of maturity levels from art to craft to science. While tacit knowledge is also part of the domain, we focus primarily on explicit knowledge because of its role in facilitating systems synthesis and analysis. However, our conceptual model takes into account that (a) explicit knowledge is not truth, it is models accepted ??? because they provide acceptable explanatory and predictive power, and subject to refinement when faced with discrepancies (b) tacit knowledge plays a role in engineering decision making. We shall see the importance of these when we discuss the application of the model to engineering practice.

In summary, the observation is that *Knowledge domains consist of related knowledge items expressed in terms of a common vocabulary. Mature domains have associated analysis theories that enable reasoning about the relationships between configurations, collections of processes enabled by the configurations, and the resulting outcomes.*

3.3 Relationships between knowledge domains: viewpoint mappings

The nature of systems is that they produce characteristics along multiple dimensions of interest, i.e. the outcomes may belong to multiple knowledge domains. In order to reason about systems, we need to reason about the relationships between knowledge domains i.e. how decisions in one area affect outcomes in other areas.

This is a central problem in software and systems engineering. Both disciplines must commonly deal with multiple knowledge domains apart from the core domains of software and systems, including application domains, resource domains (power, networking etc), technology domains and quality domains (performance, security etc), and more importantly, the interfaces and interplay among these domains. This is the reason that both disciplines need an explicit theory of the relationship between knowledge, systems and engineering.

The difficulty is that each domain has its own vocabulary. These vocabularies may (in principle) be disjoint sets. How can we work out the relationships among these different vocabularies?

If we take a closer look at the vocabulary of domains, we see that it includes not only concepts that we would consider as belonging to that domain, but also abstractions of concepts in what we would consider to be other related domains. [In reality, what has really happened is that we have noticed that these "related concepts" occur in other contexts as well, and abstracted them out to form a separate knowledge domain]. Thus, for example, knowledge about a radiotelescope dish antenna includes concepts such as a reference signal (for digitizing the received signal), power supply and network link. We think of these as abstract elements (roles, resources, operations, concerns or interfaces) that must be mapped to elements in another knowledge domain. The reasoning in a knowledge domain about how outcomes arise from structures and processes includes these abstract "external" elements as well.

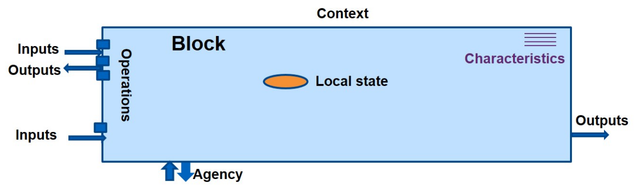
This leads us to the concept of viewpoint mappings between domains: the vocabulary of each domain includes abstractions that capture its relationship with other domains. Viewpoint mappings are generally bidirectional: each domain includes abstractions of the other, so that while the antenna requires a power supply (with associated characteristics), the power domain views the antenna as a power load. To establish the relationship between two domains, the abstractions in each domain need to be bound to system elements in the other domain.

A key observation is that these bindings occur in the context of actual systems i.e. the relationships between knowledge domains (in the general case) flows through systems. Once binding is done, it is possible to reason about the inter-relationships between domains, and flow information through the binding about how decisions in one domain affect the other.

We will discuss more about viewpoint mappings later on. The observations are that *the vocabulary of each domain includes abstractions that capture its relationships to other domains. This relationship can be expressed as viewpoint mappings. The actual relationships between knowledge domains flow through systems, and viewpoint mappings must be instantiated based on their relative roles in the system.*

3.4 System model: Block concept

Let us now turn our attention to systems, and build a conceptual model that facilitates reasoning about the relationships between systems, engineering and knowledge. We start with our concept of blocks, which are based strongly on blocks in SysML and typical systems models, with one or two variations.



A block has operations and internal state, and a set of characteristics associated with the block. Inputs and outputs to a block may be discrete or continuous. Some inputs and outputs may be distinguished as agency, which indicates that they perform higher-order functions such as control, configuration or life cycle management (monitoring, logging, test, diagnostics, upgrade etc). The concept of agency inputs may also be used to model infrastructural resources that may be required by the block (e.g. power supply, tools, shelter etc) and threats (environmental hazards, security threats etc) that do not participate directly in block functioning (except when the resource is faulty or absent), but which affect block characteristics.

A key point of departure is that each block (from an engineering and modelling point of view) includes a model of its context, a placeholder for its assumptions about the context. This brings completeness to the block for reasoning purposes, since some of the block characteristics will arise as a result of interactions between block and context (e.g. ease of use). Context consists of a collection of blocks. Also, there may be attribute relationships (e.g. influences, constraints) among the characteristics of interacting blocks. Finally, the context is considered to be open i.e. it may interact with blocks other than this block, with the possibility of (apparently) spontaneous state change.

The purpose of such a conceptual model is to facilitate more rigorous reasoning about different aspects of system behaviour and the nature of engineering practice. It is likely that in order to reason about how particular characteristics arise, we may have to consider the context to be closed, that there are no agency inputs etc. If so, these assumptions must be explicit in our model of the context, the block etc. During deployment/integration, the abstract context of the block gets bound to an actual context, at which time these assumptions can be monitored/verified.

Internally, a block consists recursively of a network of sub-blocks. Each operation of the block is realized by a process: a sequence of operations on sub-blocks (and perhaps the block itself). It is also possible that the process may include operations on (provided by) blocks in the context - this creates a *dependency*. Operations themselves have characteristics that compose through the process.

In summary, the observation is that *in order to be able to reason about the relationship between systems and engineering practice, our model of systems (blocks) must include considerations such as life cycle activities, infrastructural resources, threats, explicated assumptions about the context as part of the system model, the possibility of the context being open to external influences, the notion of deployment as binding, and a model of the nature of dependencies.*

3.5 Block compositionality

Since blocks consist of recursive compositions of sub-blocks, we need to be able to reason about how the behaviour and characteristics of a block arises from the interaction among sub-blocks, and also interactions with the context. In general, this involves the following considerations:

- Compositionality of characteristics: Given the characteristics (functions, features, services, quality characteristics) of sub-blocks, and the interaction patterns and processes that connect them, structural and process compositionality reasoning can be used (often based on patterns knowledge) to work out the relationships between the internal structure, internal processes and block outcomes. The context also plays a role in many of the outcomes, either in terms of direct participation (usage characteristics, dependencies) in outcome generation, or pathways of influence (e.g. agency inputs, resource characteristics).
- Process interplay: Compositionality reasoning often assumes that streams of input processing are mutually independent. But there may be spatial interplay: two streams of activity may both affect the state of the block, and we must make sure that the interplay preserves integrity of outcomes: the problem of concurrency. There may also be temporal interplay: in blocks with latency (delays), the processing of successive inputs even along a single input stream may interact with each other, and again engineering must ensure integrity of outcomes.
- Variety handling: Each input and state has a range of possible values, which can be divided into equivalence classes in terms of their effect on processing outcomes: different equivalence classes require different processing or produce discontinuous outcomes. Engineering, using knowledge, must identify and cover this variety space for each system element.
- Short-term dynamics: Processes transform state. Networks of processes, and processes operating continuously or iteratively, produce state trajectories. Short-term dynamics is the problem of ensuring that the state trajectories of each sub-block and the context combine to produce acceptable outcomes. This concern is recursive, so the characterization of the block should ideally include its relevant state trajectories.
- Long-term dynamics: State trajectories may cumulate over longer periods of time to create trends. There may be also be dynamics in the context with its own trends. Further, when combinations of trends result in some parameter crossing a threshold, systems undergo phase changes to new patterns of behaviour.
- Spontaneous processes may arise between system and context e.g. rusting, parameter obsolescence (for parameters intended to reflect some aspect of the context, which may change independently), that influence outcomes either over the short-term or long-term.

The observation is that *the composition of blocks must take many complex aspects into account, not all of which may be covered by the available knowledge.*

3.6 Recursive elaboration of systems

Systems are hierarchical, consisting of many levels of composition, with a network of interacting blocks at each level. Further, each block can be understood at different conceptual levels. Is there a simple way to model the nature of this elaborate structure?

We postulate that this structure can be captured in terms of three recursive elaboration operations: decomposition, dependency closure and realization, generating corresponding logical relationships among blocks:

- Decomposition is a familiar operation, generating parent-child relationships.
- Dependencies arise when the provider block is not considered part of the block generating the dependency. Dependencies need to be closed by identifying (or engineering) a block that provides the desired relationship, and eventually binding it as part of the context.
- Realization is the relationship between an abstract block and a technological block that realizes the abstraction. System elaboration may produce blocks that are expressed in terms of conceptual functions (e.g. adder, controller, braking system), which must be mapped to concrete technologies that realize the functions (e.g. adder chip, software controller, hydraulic brakes). This produces an abstraction-realization relationship between the blocks.

This recursive elaboration ends when we reach scope boundaries (dependencies that are filled by blocks in the context, and when decomposition and realization generate a need for blocks that are available or can be produced/acquired in the environment.

In summary, *we model complete systems as being generated by three recursive operations on an initial block: decomposition, dependency closure and realization.* We do not currently have any theory of completeness, to ascertain whether there are only three operations, though empirically and logically, it appears that these three are sufficient to generate a complete system from an initial block.

3.7 Relationships among knowledge domains in systems

Based on our model of blocks and how systems arise from an initial block, we can identify six kinds of relationships among knowledge domains, that generate corresponding styles of viewpoint mappings:

(1) Primary-secondary: If we look at the design of a block (e.g. a radiotelescope), we can see that it involves knowledge from multiple domains. There is a primary *functional* domain, in this case interferometry, that captures knowledge about how the primary functional outcomes (characteristics) of the block arise. There are also associated other quality characteristics, such as performance, security etc, each with their own body of knowledge. These act as *secondary* add-on domains relative to the function. Knowledge in secondary domains is expressed parametrically over abstractions of concepts in the primary domain, to facilitate synthesis of knowledge from both domains in the system. For example, security may be expressed in terms of concepts such as *resources*, *data stores* and *access operations*, and concepts in primary domains such as *signal processing equipment*, *data products* and *queries* are mapped to these abstractions for the purpose of applying security controls.

(2) Intent-Object: If we look at the relationship between a block and its inputs, the block represents a domain of activity (intent), and the inputs are the object it operates on. Activity domains (e.g. engineering, verification, risk management, sales) define the *lens* through which they want to see the objects, and the elements of any target domain on which the activity is performed must be mapped to that lens. The relationship between a block and its outputs is a process-product relationship, and this could be viewed as a subtype of the intent-object relationship.
The number of possible intent domains that can be applied to the elements of any knowledge domain is vast, probably infinite. [2] discusses how semantic web technologies can be used to formulate and support the required intent viewpoints.

(3) Parent-child: Decomposition generally creates parent-child relationships among the domains involved. Some or all of the vocabulary of the child domain is also part of the parent domain, eliminating the need for any additional viewpoint mapping.

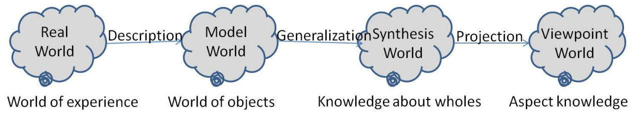
(4) Provider-consumer: Dependencies create provider-consumer relationships. In some cases, decomposition may also create provider-consumer relationships instead of parent-child relationships, if a block needs a service or capability whose realization is scoped to be within the parent block, but the knowledge domain involved is not considered part of the parent knowledge domain e.g. antenna and cooling system. Provider-consumer relationships create bidirectional viewpoint mappings as discussed earlier.

(5) Capability-Technology: Realization involves relationships between abstract capabilities and technologies that realize the capability. Capability-technology relationships involve interpretation: elements in the technology space are interpreted as implementing the concepts in the abstract capability space e.g. bit patterns in a digital adder chip are interpreted as representing numbers, and the operation performed is interpreted as addition. In the reverse direction, the physical phenomenon is abstracted and viewed as a logical concept or capability.

(6) : Whole-Aspect: While a block involves several knowledge domains, the block itself can have a knowledge domain associated with it e.g. cell biology, network routers, antennas. The relationship between the block knowledge domain and any of its viewpoint knowledge domains is that of Whole-Aspect. Vocabulary elements in the whole domain can be mapped to relevant vocabulary elements in viewpoint knowledge domains. We will see more about this relationship in the next subsection.

In summary, *each domain has associated with it a set of lenses through which it views other domains that it touches, depending on the type of relationship between them. These relationships between domains and the corresponding viewpoints are themselves part of the knowledge in each domain, though these relationships must be instantiated in the context of particular systems.* We can see this clearly if we consider the contents of textbooks on any knowledge domain, and how it handles other related knowledge domains.

3.8 The relationship between knowledge and systems



The relationships between knowledge and systems can be understood by examining an example situation, such as a candle burning on a table whose light is reflected in a mirror, and considering various related worlds of interest. The system in the real world of experience is modelled by descriptions such as the one above or a structured model in (say) SysML. A knowledge domain such as optics examines the system from a particular viewpoint, reducing the various elements to abstractions in its vocabulary e.g. a *light source* producing *beams* that *impinge* on a *reflector* and producing images. The candle, mirror, the physical environment etc can be understood from multiple viewpoints e.g. optics, chemistry, materials science, engineering, physical layout. We can and do create knowledge domains of wholes e.g. a candles knowledge domain, mirrors domain, physical environments knowledge domain, that synthesize knowledge from various applicable viewpoints and build knowledge about the whole (though the contents are intrinsically limited by the focus of observers).The system model, in order to explain, predict or achieve outcomes of interest satisfactorily, must build a fabric of reasoning that is internally consistent, consistent with the various applicable knowledge domains, and also consistent with the real world, within bounds of acceptability.

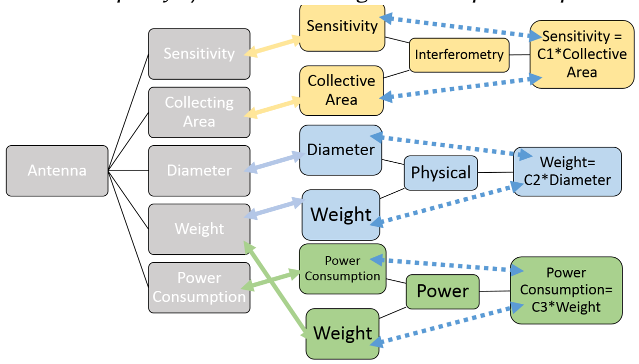
Let us now examine each of the worlds above in more detail:

- Real world: This is the world of actual experience, all others are worlds of models and description that are necessarily approximate. There is always a gap between the real world and knowledge, that must ultimately be observed and closed in the real world. In engineering, construction/implementation involves a mapping back from the world of descriptions to the real world.
- Model world: This is the world of system descriptions. Entities in this space represent specific objects (i.e. instances), which we may also abstract to types as part of this space. Binding (e.g. of roles defined by knowledge to entities that fill that role) is an operation typically performed in this world. Reasoning about outcomes is also ultimately with reference to this world.
- Viewpoint knowledge world: This is the world of foundational knowledge e.g. scientific knowledge. It is derived by abstracting from the real world, looking at it from a specific viewpoint of interest, and building causal knowledge about phenomena. Typically, such knowledge is expressed in terms of abstractions (e.g. light source). Engineering may also generate viewpoint knowledge worlds (e.g. security knowledge, insurance domain knowledge) by abstracting from experience (and/or deriving from other foundational knowledge) how to generate outcomes of interest.
- Synthesis knowledge world: This is a world of knowledge about canonical wholes, that builds on knowledge in foundational viewpoints. It should be noted that synthesis of viewpoint knowledge alone does not generate a synthesis knowledge world - it could simply generate a wider viewpoint knowledge world - whose scope of interest is wider. The distinguishing aspect of a synthesis knowledge world is that it identifies wholes of interest (which could be entities e.g. network router, or activities e.g. system engineering, or even concepts e.g. value) and knowledge in every aspect relevant to that whole is scoped to belong within the viewpoint. It defines a canonical whole and develops a consistent synthesis of knowledge from the various applicable viewpoints. Synthesis knowledge worlds are valuable because they solve at the domain level one of the most challenging problems in working with knowledge (particularly for engineering purposes) - the problem of aspect weaving. There can be extraordinary complexity in how various items of foundational knowledge interplay to generate the various characteristics and behaviours (including short and long term dynamics) of wholes, so there is great value in working through the complexity and generating a body of knowledge e.g. about antennas or network routers. Science also generates synthesis knowledge worlds e.g. cell biology.

  There is often a many-to-many mapping or modified mapping between wholes in the systems model world and the canonical wholes in the synthesis knowledge world, necessitating additional effort to ensure correct mapping and application of knowledge. It should also be noted that what constitutes a relevant synthesis viewpoint depends on the system-of-interest, and the type of wholes it is interested in.

The summary is that *knowledge often belongs to viewpoints. Working out how viewpoint knowledge comes together to generate the characteristics of wholes is often extremely challenging. We typically solve this problem by developing synthesis knowledge domains that (largely) solve the aspect weaving problem at the domain knowledge level.*

3.8.1 An example of synthesis knowledge and viewpoint mappings.



The diagram above shows a small example of the synthesis of knowledge about dish antennas based on viewpoint mappings. It shows snippets of three applicable viewpoints: interferometry, physical and power, and how this knowledge is brought together to create domain knowledge about dish antennas. It also illustrates the foundations of the viewpoint mapping concept: viewpoints are generated by abstracting out different aspects of a whole, so viewpoint mappings in general involve an abstraction step, and the binding of the relationship between viewpoints must always flow through the whole that generates the viewpoints.

Of course, the example shown is trivially simple and does not get into the true complexities of aspect weaving and viewpoint mapping e.g. when the concepts involved in different viewpoints do not line up perfectly. However, it illustrates the key point that we want to make, that especially to resolve complex situations, it is necessary to work through a model of the whole that generates the viewpoints.

The above subsection needs to be significantly rewritten. Need more clarity on messaging, and write to bring out the message. Perhaps too many asides.

Add power system, and link it to Telescope. Convey more strongly about working through wholes. Simplify synthesis domain message.

3.9 Engineering: Problem Formulation

The need for engineering is triggered either by a situation that needs to be transformed (gaps to be closed or opportunities to be pursued) or by stakeholder goals to be met.

In our conceptual model, we see situations as a system with characteristics. In order to change situation characteristics, we need a problem formulation activity that analyzes how the internal structure of this system needs to change in order to achieve the desired modified characteristics. The result of this problem formulation is the specifications for entities that need to be added or modified, to obtain this structural change.

If the trigger is from stakeholders, the problem formulation activity needs to establish the processes and structures needed to achieve their goals. Stakeholders can be viewed as one of the blocks in the context of the system-of-interest. Some of these structures and process steps will be provided/enabled by other blocks in the context, but some may require contributions from the system-of-interest. The result of problem formulation is the specifications that the system-of-interest must satisfy in order to make those contributions.

Incidentally, the value contributed by the system-of-interest can be determined by tracing the above reasoning chain in the opposite direction: establish the contribution made by the system-of-interest to the goal accomplishment process of stakeholders, then determine the effect of these contributions on the overall goals. This establishes objective value delivered, from which perceived value can be determined by adding contextual modifiers reflecting particular stakeholders' needs, preferences and perceptions.

In both cases, engineering problem formulation involves reasoning about the context as a larger system (supersystem) of which the eventual system-of-interest forms a part. We can identify supersystems for a given system-of-interest along multiple dimensions (e.g. operational, financial, engineering, regulatory etc). Triggering situations or stakeholder goals may also span these multiple dimensions (which may interact with each other). Ideally, engineering should formulate models of these supersystems (context blocks), and reason about how the introduction of new or modified entities with particular specifications will lead to the targeted situational changes or stakeholder goals, including analysis of the transitional, short-term and long-term dynamics involved.

The summary is that *engineering problem formulation involves reasoning about the systemic logic of how the specifications of the system-of-interest will enable the desired changes in contextual outcomes.*

3.10 Engineering: Steps needed

Based on the conceptual model, we can see that in order to transform the situation or enable stakeholder goal achievement, engineering needs to do the following:

- Formulate the problem in terms of outcomes to be delivered by a system-of-interest.
- Synthesize a conceptual solution to the problem consistent with relevant knowledge. Elaborate the solution by recursively performing the activities of decomposition, dependency closure and realization, until we arrive at blocks that are available in the context or can be produced.
- Translate the resulting description into a real-world implementation.
- Bind the implementation to the target operating context.
- Determine whether the desired real-world outcomes have been achieved, and if not, close the gaps.
-

The observation is that *while engineering modelling and reasoning happens in the description world, ultimately engineering is about making changes in the real world, and its activities must include identifying and closing any gaps between the real world and the conceptual and knowledge worlds.*

3.11 Engineering: Ecosystem

Engineering itself is an intent domain, with associated knowledge about the activities to be performed: an elaboration of the steps outlined above. But this is a viewpoint knowledge domain - knowledge about engineering from the viewpoint of achieving one outcome: product delivery. To ensure that engineering itself is successful, we must look at it as a whole, and apply knowledge from the corresponding synthesis knowledge domain: the set of all concerns and outcomes associated with engineering, the influencers of these outcomes, the context within which engineering is performed, and knowledge about how to achieve the desired combination of outcomes in the particular context.

This body of knowledge brings in the entire set of ecosystem concerns: quality management, planning, resources, projects and their management, strategic concerns, reuse etc. In other words, we can work out the need for different parts of the engineering ecosystem, by applying the conceptual model: identify the whole involved, the collection of concerns associated with the whole, and synthesizing knowledge from the relevant viewpoints.

Ecosystem thinking must also be applied to the product itself, so that engineering deliverables (in principle) include not only a core solution that achieves defined goals, but an associated ecosystem (including people processes) for addressing associated operational, life cycle and usage concerns, including monitoring, control and management of the solution, diagnostics, maintenance, upgrades and other life cycle concerns, usage help, feedback and continuous closure of the gaps between solution and context, and sustainability of the solution given the dynamics of the context.

The observation is that *the principle of looking at wholes and structuring knowledge as synthesis domains that draw upon viewpoint domains allows us to modularize knowledge, so that, for example, software or systems engineering need not reinvent knowledge from project management space or quality management space, but frame it as a problem of identifying and synthesizing relevant knowledge from viewpoint spaces. Applying this principle to the product expands the scope of engineering to the solution ecosystem.*

3.12 Engineering: Relationship to knowledge and systems, and challenges

Engineering is a knowledge-centric activity. Knowledge plays a role at multiple levels:

- As stated above, engineering itself is a synthesis knowledge domain, that plays the role of an intent domain relative to the product.
- Both engineering decision-making and engineering reasoning rely on knowledge. The knowledge may be formal or informal, and exist either as viewpoint knowledge, synthesis knowledge, or simply practice rules not considered part of any particular domain, plus of course tacit knowledge.
- Knowledge has a role to play in the characterization of wholes: identifying all the relevant concerns that need to be addressed for each element of the solution, and at all levels of the solution.
- Many of the activities in engineering, including problem formulation, context understanding, technology selection and engineering ecosystem design itself, require situational

knowledge: information about worlds of interest. Engineering decision-making uses situational knowledge, about which concerns are important, about various decision influencers (such as people competences) etc.
- Finally, engineering also generates and refines knowledge in the various domains involved, and about gaps between the real world and the model world. Explication of the relationships between knowledge and engineering can help with harvesting this knowledge.

The conceptual model points out some of the challenges involved in engineering:

- Knowledge often exists in viewpoints. Identification and synthesis of all relevant knowledge in the context of each of the wholes in the system is the job of the engineer. Even if there is a synthesis domain corresponding to a particular whole, problem, strategic and contextual considerations often result in mappings being needed between the system wholes and canonical wholes, sometimes even many-to-many mappings, that engineers must work through to have a firm basis for decision-making and reasoning.
- Knowledge available is often incomplete with respect to the wide range of systems concerns that must be addressed.
- Context is, in general, an open world, but compositionality reasoning requires that we make assumptions about the context. Ideally, these need to be made explicit, and deployment (binding to actual contexts) must include validation of these assumptions.
- Engineering can be viewed as an enormous network of decisions, spanning problem, solution and context; product and engineering ecosystem; hierarchical levels of the system; dimensions of outcomes for each block; and range of compositionality concerns for each block. Successful engineering requires achieving mutual consistency across this entire network (within tolerances!).
- Knowledge and reasoning are at the level of description and models, but engineering is responsible for outcomes in the real world.

The summary is that *Engineering is a knowledge-centric activity, with both domain knowledge and situational knowledge needing to flow into both decision-making and reasoning at all levels. Engineering can be viewed as an enormous network of decisions with mutual consistency relationships.*

3.13 Summary of the conceptual model

We can get an overview of the conceptual model by collecting and summarizing the observations from each subsection:

- Systems knowledge is primarily in the form of patterns that express relationships between structures, processes and outcomes, qualified by context.
- Knowledge domains consist of related knowledge items expressed over a common vocabulary. Mature domains may have associated synthesis and analysis theories.
- The vocabulary of each domain includes abstractions that capture its relationships to other domains. This relationship can be expressed as viewpoint mappings. The actual relationships between knowledge domains flow through systems, and viewpoint mappings must be instantiated based on their relative roles in the system.
- We can model systems using the concept of blocks. These blocks must be formulated in a way that can model the range of concerns and complexities encountered in engineering.
- The composition of blocks must take many complex aspects into account, not all of which may be covered by the available knowledge.
- We model complete systems as being generated by three recursive operations on an initial block: decomposition, dependency closure and realization.
- There are different kinds of viewpoint mappings, reflecting different relationships between knowledge domains in the context of a system. Viewpoint mappings are themselves part of knowledge in each domain - the lenses through which it views other domains that it touches.
- We can understand the relationship between systems and knowledge in terms of four worlds: real world, systems model world, viewpoint knowledge world and synthesis knowledge world. Synthesis knowledge worlds are extremely useful because they address the aspect weaving problem of working out how knowledge from different domains combines in the context of particular wholes.
- Engineering is triggered by gaps or opportunities in situations or by stakeholder goals. Engineering problem formulation involves reasoning about the systemic logic of how the specifications of the system-of-interest will enable the desired changes in contextual outcomes.
- Engineering decision-making and reasoning occur in the system model world, and it must include activities to identify and close gaps with the real world.
- Engineering requires an engineering ecosystem, and its scope should include the product ecosystem. The need for these becomes evident from the principle that systems act as a whole, and must be dealt with as wholes. We can use the approach of synthesis domains and viewpoint domains so that knowledge in these areas can build on foundational knowledge relating to each of the ecosystem concern spaces.
- Engineering is a knowledge-centric activity, with both domain knowledge and situational knowledge needing to flow into both decision-making and reasoning at all levels. Engineering can be viewed as an enormous network of decisions with mutual consistency relationships.

The conceptual model is that knowledge domains are often based on viewpoints, which can be synthesized in the context of canonical wholes to generate synthesis knowledge domains. We need viewpoint mappings to connect knowledge domains, which needs to be done in the context of systems. Thus the connection between knowledge and systems goes through wholes.

Systems can be viewed as a network of blocks that collectively produce outcomes of interest, with the network being generated by three recursive operations: decomposition, dependency closure and realization. In general, the problem of block compositionality involves several issues to be considered.

Engineering involves building mutual consistency among a network of decisions ranging from stakeholder or situational goals to problem formulation to solution concept to its realization in the real world to identifying and closing gaps in the real world solution. The network also spans the engineering and product ecosystem and context, and all levels in the solution. The decisions and reasoning at each node in the network is informed by situational and domain knowledge, both formal and informal. Synthesis viewpoint domains can help with the challenge of bringing together knowledge from multiple domains in a consistent way.

4 MODEL VALIDATION

This conceptual model was developed by explicating and building on the explicit knowledge, tacit understanding and intuitions prevalent in the systems and systems/software engineering space. One of its major goals is to provide a framework for reasoning about engineering practice. In this section, we attempt to validate it by exploring its power to explain the basis for widely accepted current practices, that are currently viewed as being empirical.

First, we look at how the steps identified by the conceptual model (section 3.10) map to current process areas, particularly the waterfall model. The following table explores this mapping:

Processes	**Conceptual model**
Mission analysis	Identifying blocks to be added or changed to achieve situational goals
Stakeholder needs definition	Contributions desired from system-of-interest
Requirements specification	Specifications for system-of-interest
Architecture	– none – (not addressed)
Design	Network of blocks
Implementation	Implement in real world
Integration	Binding together of sub-blocks to form larger blocks
Verification	See below
Deployment	Binding to operational context
Validation	Identifying and closing gaps
Operations	Usage

The steps in the conceptual model map reasonably well to the practice, providing a conceptual perspective on the role of several development life cycle processes. While a detailed discussion is beyond the scope of this paper, the value of such a conceptual foundation is that it can be used to work out the responsibilities and issues to be addressed by each process. Further, it enables the framing the deliverable of each process as a model (or real world artifact) with mutual consistency relationships.

Architecture is missing because the conceptual model has not been elaborated to expose the concern being addressed by architecture. It was mentioned earlier in the discussion on block compositionality is that one of the concerns is the impact of long-term dynamics. This is one of the concern being addressed by architecture. Architecture is also used to address the concern of change stability (small changes in specifications do not cause large perturbations in solution) and commonality across variants (e.g. product line) - again concerns that we did not explicitly bring out in the model. We do have plans to extend the model to address architecture.

The other life cycle processes (planning, quality management, configuration management etc) belong to the ecosystem. While it is possible to reason about those also using the conceptual model, that is beyond the scope of this paper.

Life cycle processes is a relatively straightforward area to address. Let us now consider a more challenging problem: why do we need so many levels of verification: analysis, verification, validation, monitoring operational gaps? Could a sufficiently strong synthesis theory or proof of correctness eliminate these?

A synthesis method selects and synthesis patterns from multiple knowledge domains to address the collection of desired outcomes, ideally with proofs that show how the synthesis is consistent with applicable knowledge in each domain. Analysis methods project the solution schema onto viewpoints, and use analysis theory in each viewpoint to show that relevant outcomes are achieved by the schema. Verification binds the implemented solution with simulated or real operational contexts, and observes whether the outcomes match what is desired. Validation determines whether the actual outcomes are consistent with stakeholder goals and desired situational characteristics. Finally, during operations, monitoring checks whether the system is continuing to produce the desired outcomes. If we are able to construct a powerful synthesis methods with proofs that the solution schema produces desired outcomes, will all the subsequent activities be made redundant?

The conceptual model is able to answer this question, by indicating the additional concerns and issues addressed by each subsequent level of activity, as shown in the table below:

Activities	**Issues and concerns addressed**
Synthesis	Compositionality of selected patterns, consistency with explicit knowledge.
Analysis	Spontaneous processes and interactions (not part of selected patterns), tacit knowledge (since analysis operates over entire configuration, including implicit patterns), gaps in explicit patterns knowledge.
Verification	Gaps in construction, gaps in knowledge, gaps in reasoning. Gaps in context assumptions across blocks: behaviour of a block does not match assumptions of another block with which it interacts. System verification also addresses gaps between assumed and actual context, and gaps between real world and model world.
Validation	Gaps in problem formulation. Dynamics of interaction between system and context, including openness of context. Subjective aspects, such as stakeholder preferences and perceptions.
Monitoring	Long-term dynamics of system and context. Effects of system, context and stakeholder evolution

The conceptual model gives us the vocabulary and insights to reason about the role of each activity.

The model can also be used to explain various engineering phenomena and issues. For example, the challenges of interoperability can be understood in terms of interpretation relationships: while two interacting blocks may share common vocabulary elements at the conceptual level, enabling interaction, when we bind technologies to each block, we must make sure that the vocabulary/elements in the two technological domains (that each interpret/implement the vocabulary elements of the interface) are mutually compatible. This must be true for the successive levels of technological bindings, all the way down to the connectors that facilitate the interaction.

Apart from validating the conceptual model in terms of its explanatory power (and perhaps even predictive power) for engineering practice and associated issues and observations, we have also examined it in the light of actual projects. Discussions have indicated that for several of the flagship projects in our organization, their key innovations in the areas of how knowledge domains relate to each other can be understood in terms of viewpoint mapping concepts. We have also created a proof-of-concept that successfully applies the viewpoint mapping theory to the problem of creating an integrated model of the GMRT radiotelescope, that enables what-if queries across what would normally be separate knowledge domains e.g. how a change in telescope sensitivity requirements affects power consumption.

5 POTENTIAL UTILITY OF THE MODEL

Currently, we see a couple of major uses for such a conceptual model. First, it can help us reason about engineering practice, converting it from a purely empirical space to one that has some underlying theory about its nature. Of course, the current model is only an initial strawman version, and will require considerable refinement and socialization before it can assume the dimensions of a foundational framework, but at first glance it seems to have enough soundness to envisage such a possibility.

It can also inform practice in the area of modelling and knowledge management. A key insight provided by the model is that we need to build knowledge about the nature of wholes, in order to facilitate engineering. We need to work on capturing knowledge in canonical synthesis knowledge domains, that build on foundational knowledge and indicate how viewpoints come together. Standardization bodies in various spaces are already doing this, but such theory can inform and catalyze these efforts. Moreover, system models (descriptions) should indicate the mapping between system elements and corresponding canonical wholes, so that the semantic basis for viewpoint construction and reasoning is part of the system model. This will also enable integration of tools across viewpoints, since the necessary semantic basis and inter-relationships between viewpoints would now be part of the system model.

6 CONCLUSION

We have presented a strawman conceptual model that has been incrementally developed by inquiry into the nature of knowledge in relationship to systems and their engineering. Preliminary explorations of the explanatory power of the model are giving us some initial confidence in its validity. More specifically, it appears that the line of inquiry and approach are fundamentally sound, though it is quite likely that many of the concepts need to be evolved and expanded.

We are ourselves aware of only a very limited part of the knowledge in this area, and would welcome inputs from the community to help with refining the model and validating it. We plan to work with the INCOSE systems science community on taking this forward, but would welcome inputs and assistance from other communities as well. Our eventual goal is to develop conceptual foundations for software and systems engineering practice, and to enhance system modelling approaches such as SysML with conceptual foundations and additional capabilities that express the ontological basis of the model to facilitate reasoning.

7 ACKNOWLEDGEMENTS

The ideas described in this paper have evolved over more than a decade in TCS Research, at the Business Systems Cybernetics Centre and the Systems Research Lab. Many of our colleagues in these and other TCS Research Labs have contributed significantly to the understanding. Niruj Mohan Ramanujam and Rajiv Banerjee of NCRA-TIFR were instrumental in working through the area of viewpoint mapping in the context of the GMRT radiotelescope. Our discussions with members of various international communities, including INCOSE and the ISO SC7 standards body, were also instrumental in shaping the ideas.

REFERENCES

[1] Albert Benveniste, BenoÃőt Caillaud, Dejan Nickovic, Roberto Passerone, Jean-Baptiste Raclet, Philipp Reinkemeier, Alberto Sangiovanni-Vincentelli, Werner Damm, Thomas Henzinger, and Kim G. Larsen. 2012. *Contracts for System Design.* Technical Report 8147. INRIA. 0–0 pages.

[2] M. Blackburn and P. Denno. 2015. Using Semantic Web Technologies for Integrating Domain Specific Modeling and Analytical Tools. In *Complex Adaptive Systems Conference.*

[3] Jamshid Gharajedaghi. 2012. *Systems Thinking: Managing Chaos and Complexity: A Platform for Designing Business Architecture* (3 ed.). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[4] Kesav V. Nori and N. Swaminathan. 2006. A framework for software product engineering. In *13th Asia-Pacific Software Engineering Conference APSEC 2006.*

[5] Uri Shani, Shmuela Jacobs, Niva Wengrowicz, and Dov Dori. [n. d.]. Engaging ontologies to break MBSE tools boundaries through semantic mediation. In *2016 Conference on Systems Engineering Research.*

[6] K. Viswanath. 2008. *An Introduction to Mathematical Computer Science.* The Universities Press, Hyderabad, India.